Integral de Riemann. Integración numérica.

Práctica 5 (Específica de la asignatura de Cálculo Matemático en E.U.A.T.)

(Práctica elaborada a partir de las realizadas por los profesores Jesús Montejo, Juanjo M. Nieto y Óscar Sánchez.)

Introducción

En esta práctica veremos como calcular integrales definidas, primitivas de funciones dadas y aproximar integrales definidas con *Mathematica.* Para ello usaremos algunas funciones especiales que tiene *Mathematica* para realizar este tipo de tareas.

Motivación geométrica de la integral definida

El problema del que partimos es el siguiente: tenemos definida una función acotada *y=f(x)*, en el intervalo cerrado y acotado *[a,b]*, y queremos calcular el área que encierra la gráfica de esta función y el eje de abcisas. Suponemos que se miden áreas positivas si la función está por encima del eje de abcisas (eje *OX*) y áreas negativas si la gráfica de la función se encuentra por debajo del mismo.

```
Clear["Global`*"]
a = 0; b = 1;
f[x_] := x^2 + 1
```

Representemos la región plana a la que queremos calcularle su área.Para ello utilizamos la orden **Plot** con distintas opciones. Con la opción **PlotRange** especificamos el rango de la variable dependiente *y*. La opción **Epilog**, así como **Prolog**, nos permite insertar nuevos elementos gráficos a la vez que se representa la gráfica de la función. En este caso hemos dibujado un trozo de linea recta que cierra la región delimitada por nuestra función y el eje *OX*. También hemos puesto etiqueta a los ejes con la opción **AxesLabel** y al propio dibujo con **PlotLabel**.

```
region = Plot[f[x], {x, a, b}, PlotRange → {0, f[b]},
   Epilog → {Line[{{b, 0}, {b, f[b]}}]},
   AxesLabel → {" x", "y"}, PlotLabel → "y = f[x]"];
```

Para mostrar las ideas empezamos con un caso fácil en el que se divide el intervalo en sólo dos subintervalos iguales. Podemos considerar entonces rectángulos cuya base es la mitad del intervalo y cuya altura es el extremo derecho de cada subintervalo. Por tanto, si nuestra función es monótona creciente, obtenemos una aproximación por exceso del área que queremos calcular.

```
Clear[n, h, c]
n = 2;
h = N[(b - a)/n];
c = a + h;
```

Utilizamos la gráfica predefinida **Rectangle** para dibujar rectángulos paralelos a los ejes de coordenadas, sin más que especificar dos de sus vértices en diagonal. La opción **GrayLevel** nos permite indicar el nivel de gris utilizado para colorear dichos rectángulos. Todo ello se puede incluir en un mismo gráfico junto con la región determinada por nuestra función gracias a la orden **Show** junto con la opción **Prolog**, que actúa de forma parecida a la opción **Epilog**.

```
Show[region, Prolog → {{GrayLevel[.8],
     Rectangle[{a, 0}, {c, f[c]}], Rectangle[{c, 0}, {b, f[b]}]}}];
```

Para calcular el área que determinan estos rectángulos; bastará multiplicar la longitud de la base de ambos, *h*, por la altura de cada una de ellos y sumar.

```
h f[c] + h f[b]
```

Si por el contrario, tomamos como altura de cada uno de los rectángulos, el valor de la función en el extremo de la izquierda, para una función creciente, obtendremos una aproximación por defecto del área que queremos calcular.

```
Show[region, Prolog → {{GrayLevel[.8],
     Rectangle[{a, 0}, {c, f[a]}], Rectangle[{c, 0}, {b, f[c]}]}}];
```

```
h f[a] + h f[c]
```

Es claro que estas aproximaciones, con tan solo dos subintervalos, son muy "burdas" como para considerar que son los valores que miden el área que queremos calcular. De hecho sólo cabe considerarlas como valores indicativos entre los que debe de estar el verdadero. Parece evidente que cuantos más subintervalos tomemos, es decir, cuanto mayor sea *n*, estas aproximaciones, tanto por defecto como por exceso, serán cada vez mejores. Comprobemos la validez de esta idea.

Supongamos que tenemos dividido el intervalo cerrado y acotado *[a,b]* en *n* partes iguales. Cada una de estas partes se corresponderá con el subintervalo $[x_{i-1},x_i]$, donde $x_i = a + ih$, con $i=0,1,...,n$, y $h=(b-a)/n$.

```
Clear[n, h, i, x]
n = 4;
h = N[(b - a)/n];
x[i_] := x[i] = a + i h
```

```
Show[region, Prolog → {{GrayLevel[.8],
     Rectangle[{a, 0}, {x[1], f[x[1]]}], Rectangle[{x[1], 0}, {x[2], f[x[2]]}],
     Rectangle[{x[2], 0}, {x[3], f[x[3]]}], Rectangle[{x[3], 0}, {x[4], f[x[4]]}]}}];
```

Ahora el área de estos rectángulos será

```
h f[x[1]] + h f[x[2]] + h f[x[3]] + h f[x[4]]
```

Esta es la fórmula de integración aproximada conocida por el nombre de "fórmula del rectángulo derecho", pues se aproxima el valor de la integral por la suma de las áreas de rectángulos cuya base es siempre *h* y cuya altura se va calculando, para cada subintervalo $[x_{i-1},x_i]$, como el valor de la función en el extremo derecho, $f(x_i)$. Su fórmula

general, para un *n* cualquiera es la siguiente.

Sum[f[x[i]]*h,{i,1,n}].

```
Sum[f[x[i]] * h, {i, 1, n}]
```

También podemos hacer lo mismo tomando como altura de cada rectángulo el valor de la función en el extremo de la izquierda, $f(x_{i-1})$, y obtenemos la conocida fórmula de integración aproximada del rectángulo izquierdo.

Sum[f[x[i-1]]*h,{i,1,n}].

```
Show[region, Prolog → {{GrayLevel[.8],
     Rectangle[{a, 0}, {x[1], f[x[0]]}], Rectangle[{x[1], 0}, {x[2], f[x[1]]}],
     Rectangle[{x[2], 0}, {x[3], f[x[2]]}], Rectangle[{x[3], 0}, {x[4], f[x[3]]}]}}];
```

Utilicemos pues la expresión de la fórmula del rectángulo izquierdo para calcular el área total de los correspondientes rectángulos.

```
Sum[f[x[i - 1]] * h, {i, 1, n}]
```

Como en nuestro ejemplo la función es monótona creciente, entonces cualquier suma usando el valor de la función a la derecha de los subintervalos nos proporciona un valor por exceso del área que queremos calcular, mientras que cualquier suma utilizando la fórmula del rectángulo izquierdo nos da un valor por defecto de dicha área. En general a este tipo de aproximaciones por exceso se les denominará sumas superiores. A las aproximaciones por defecto les llamaremos sumas inferiores.

Resulta evidente que cualquier suma superior construida de esta forma será siempre mayor que cualquier suma inferior. Por tanto, el conjunto de todas las sumas inferiores estará siempre acotado superiormente por cualquiera de las sumas superiores. Análogamente, el conjunto de todas las sumas superiores va a estar acotado inferiormente por cualquiera de las sumas inferiores. Se puede entonces tomar el supremo de uno de los conjuntos y el infimo del otro, teniendose así las denominadas integral inferior e integral superior de la función en el intervalo considerado. El valor exacto del área de nuestra región debe de encontrarse comprendido entre los valores de la integral inferior y la superior de la función.

Si ambas integrales, la inferior y la superior, coinciden, entonces se dice que la función es integrable según Riemann en dicho intervalo. En este caso el valor de la integral debe de ser forzosamente el área de la región limitada por la gráfica de la función y el eje de abcisas entre los extremos del intervalo de integración.

Resultados de las sumas de Riemann correspondientes a las fórmulas aproximadas de los rectángulos

A continuación veremos los resultados numéricos aproximados que se obtienen al efectuar las sumas de Riemann correspondientes a las fórmulas de los rectángulos, izquierdo y derecho respectivamente, para diferentes valores de *n*. Estos resultados se dan en una tabla utilizando la sentencia **TableForm**.

```
TableForm[Table[{n, Sum[f[a + ((i - 1) (b - a))/n] N[(b - a)/n], {i, 1, n}],
                   Sum[f[a + (i (b - a))/n] N[(b - a)/n], {i, 1, n}]},
  {n, 2, 10, 2}], TableHeadings → {None,
   {"numero de\nrectangulos\n", "rectang. izdo.\n", "rectang. dcho.\n"}}]
```

Observamos que, como era de esperar, al aumentar *n* los valores por defecto, que miden aproximadamente el área, van aumentando, mientras que los valores por exceso van disminuyendo cada vez más. Esto se explica ya que, al aumentar el número de rectángulos considerados, éstos cubren con mucha más precisión la región a la que queremos calcularle el área y, por tanto, el error que cometemos será cada vez menor.

No obstante, siempre tenemos la opción de calcular el valor exacto del área. Para ello evaluamos la correspondiente integral mediante el uso de alguna primitiva de la función. Si ello no fuera posible, podemos utilizar algún otro método de integración numérica más preciso que el de los rectángulos. Los más usados son el método del trapecio y el método de Simpson, que se describen más adelante. Por supuesto hay otros métodos mucho más sofisticados.

```
Integrate[f[x], {x, a, b}]
NIntegrate[f[x], {x, a, b}]
```

Integral indefinida. Función de acumulación

Recordamos las definiciones y resultados fundamentales. Dada una función continua *f(x)* en un intervalo *[a,b]*,

- decimos que *F* es una primitiva de *f* cuando *F'(x) = f(x)* para cada *x* en *[a,b]*;
- la función de acumulación: $F(x) = \int_a^x f(t)\,dt$ es una primitiva de *f* (Teorema Fundamental del Cálculo Integral);
- Si *G* es una primitiva cualquiera de *f*, entonces $\int_a^b f(x)\,dx = G(b) - G(a)$ (Regla de Barrow).

Veámos gráficamente cómo es la función de acumulación. Representamos (en negro) la función $f(x) = \frac{x}{6}\,sen(x)$ en el intervalo $[0,4\pi]$ y su función de acumulación *F*(*x*) (en azul). En la evolución vemos cómo al barrer áreas positivas (en verde) *F* crece, mientras que las "áreas negativas" (en rojo) al acumularse van restando y hacen decrecer a *F*. Por ello a *F* se le llama también función de acumulacion de área. Para poder ejecutar el programa diseñado, necesitamos previamente cargar un paquete auxiliar.

```
<< Graphics`FilledPlot`

areaMovie[func_, {x_, a_, b_}, {ymin_, ymax_}, frames_] :=
 Module[{curve, shade, nada = Graphics[{GrayLevel[1], Point[{2 b, 0}]}], ⌘graph, f},
  f[t_] = func /. x → t; Do[curve = Plot[f[t], {t, a, b}, DisplayFunction → Identity];
   shade = If[a < x, FilledPlot[{Max[f[t], 0], Min[f[t], 0]},
      {t, a, x}, Fills → {{{1, Axis}, Hue[.42]}, {{2, Axis}, Hue[0]}},
      Curves → Front, DisplayFunction → Identity], nada]; ⌘graph = If[a < x,
     Plot[NIntegrate[f[s], {s, a, t}, AccuracyGoal → 3], {t, a, x}, PlotDivision → 4,
      PlotStyle → {Hue[.7], Thickness[.005]}, DisplayFunction → Identity], nada];
   Show[curve, shade, ⌘graph, Graphics[{Hue[.7], PointSize[.015],
      Point[{x, NIntegrate[f[s], {s, a, x}, AccuracyGoal -> 3]}]}],
    Graphics[{GrayLevel[.5], Line[{{x, ymin}, {x, ymax}}]}],
    PlotRange → {{a, b}, {ymin, ymax}},
    DisplayFunction → $DisplayFunction], {x, a, b, (b - a) / (frames - 1)}]]

areaMovie[x / 6 Sin[x], {x, 0, 4 π}, {-3, 3}, 20]
```

Ejercicios (Primera parte)

1.- Comprueba que pasa ahora si introducimos una función negativa, por ejemplo *f(x)=-x^2* en el intervalo *[0,1]*. (Para ello sal de la práctica sin grabar nada y vuelve a entrar para ejecutar de nuevo las sentencias cambiando apropiadamente la definición de la nueva función.) ¿Cómo se explican los resultados que se obtienen?

2.- Repite el mismo proceso que el seguido para la función del ejemplo, con una función del tipo *f(x)=x^3+ds* en el intervalo *[d1,dm]* (donde *d1* es el primer dígito de tu DNI y *ds*, *dm* son la suma y la media de todos los dígitos)

3.- Comprueba ahora que pasa cuando consideramos una función decreciente en lugar de una creciente; toma por ejemplo *f(x)=1/x* en el intervalo *[d1,dm]*. ¿Qué diferencia fundamental encuentras entre este caso y el del ejemplo inicial? (Sugerencia: para visualizar mejor la gráfica de una función positiva y decreciente, cambia la opción **PlotRange->{0,f[a]}**.)

Dos comandos nuevos: Integrate y NIntegrate

■ Integrate

Si *f* es una función, la orden **Integrate[f[x],x]** da como resultado una primitiva (a veces se llama integral indefinida) de la función *f*. **Integrate** requiere dos argumentos: la función que queremos integrar y la variable de la que depende.

```
Integrate[x^2, x]
```

```
Integrate[1 / (x^2 + 1), x]
```

En lugar de escribir **Integrate** podemos usar el símbolo de integral que aparece en la paleta de *Mathematica* (está en File/Palettes/BasicInput). El resultado es el mismo.

$$\int x^2 \, dx$$

Integrate realiza cálculos simbólicos, es decir, el resultado es exacto (no se trata de aproximaciones numéricas). Por ejemplo, puede aparecer Pi en el resultado en lugar de 3.14159... .

Hay que tener en cuenta que no siempre es posible calcular una primitiva de una función con funciones elementales. En estos casos *Mathematica* no sabrá dar una respuesta o la dará usando funciones menos corrientes.

```
Integrate[Sin[x] / x, x]
```

```
Integrate[Sqrt[1 + (Cos[x])^2], x]
```

```
Integrate[Exp[x^2], x]
```

Cuando la integral que buscamos tiene cierta complicación (por ejemplo, incluye senos, cosenos y raíces), lo más común es que *Mathematica* no pueda dar una expresión de la primitiva (en la mayoría de los casos porque la integral no puede calcularse en términos de funciones conocidas).

```
Integrate[Sin[x] / (Sqrt[Cos[x]^2 + 3 + x]), x]
```

Integrate puede calcular también integrales definidas. Para eso es necesario especificar los límites de integración entre llaves, además de la variable en la que se integra.

```
Integrate[Sin[x], {x, 1, 2}]
```

Si la función que escribimos no es integrable, **Integrate** puede descubrir el error.

```
Integrate[1 / x, {x, 0, 1}]
```

Por supuesto, una vez que obtenemos un resultado exacto podemos calcular aproximaciones numéricas usando la orden **N**.

```
integral1 = Integrate[1 + Sqrt[x], {x, 10, 20}]
```

```
N[integral1]
```

Por último, hemos de comentar que **Integrate** también permite trabajar con integrales de Riemann impropias.

```
Integrate[1 / (x^4 + 1), {x, -Infinity, Infinity}]
```

```
Integrate[4 * x^2 Exp[x^3], {x, -Infinity, 2}]
```

■ NIntegrate

Como dijimos antes, a veces no es posible calcular explícitamente una primitiva de una función. Además, si lo que nos interesa es el valor numérico de cierta integral definida, normalmente es mucho más rápido usar la orden **NIntegrate** que calcular la integral con **Integrate**. La orden **NIntegrate** se usa igual que **Integrate**, con la diferencia de que sólo calcula integrales definidas y da como resultado su valor numérico.

```
NIntegrate[1 + Sqrt[x], {x, 10, 20}]
```

Podemos observar la diferencia de tiempo entre calcular una integral complicada de forma simbólica y calcular una aproximación. Si hacemos la siguiente integral numéricamente con **NIntegrate** el resultado es rápido.

```
integraldificil =
 NIntegrate[(Cos[x] + Log[x] + x^5) / (Sqrt[Cos[x]^2 + 3 + Exp[x]]), {x, 3, 4}]
```

Si intentamos calcularla simbólicamente, *Mathematica* pierde mucho tiempo sólo para comprobar que no puede hacer la integral.

```
integraldificil2 =
 Integrate[(Cos[x] + Log[x] + x^5) / (Sqrt[Cos[x]^2 + 3 + Exp[x]]), {x, 3, 4}]
```

```
N[integraldificil2]
```

A veces el comando **NIntegrate** "necesita" ayuda. Veamos un ejemplo.

```
NIntegrate[(1 / x) * Sin[1 / x], {x, .001, 1}]
```

Mathematica informa de que el resultado no es fiable. En este caso es debido a que la función oscila demasiado. Cuando esto ocurre podemos intentar aumentar el número de iteraciones con la opción **MaxRecursion**, aumentar la precisión de los cálculos con la opción **WorkingPrecision** o bien usar otras opciones (que debes consultar en la ayuda de *Mathematica*).

```
NIntegrate[(1 / x) * Sin[1 / x], {x, .001, 1}, MaxRecursion → 40]
```

Métodos numéricos de integración

Los métodos numéricos de integración son formas de calcular el valor numérico de una integral definida. Desde luego, *Mathematica* (y muchos otros programas) pueden calcularlo directamente porque internamente usan uno de estos métodos numéricos de integración. Entender cómo funcionan estos métodos ayuda a entender el significado de la integral definida. De hecho, la definición de la integral de Riemann contiene implícitamente una forma de calcularla, tal como hemos visto al rpincipio de esta práctica.

A continuación vamos a repetir el proceso deuna manera más sistemática. Empezamos definiendo *f(x)=sen(x)+1* de forma que sea una función numérica. Para ello incluimos el comando **N**.

```
f[x_] := N[Sin[x] + 1]
```

Para aproximar su integral de Riemann en un intervalo *[a,b]* podemos partir el intervalo en trozos pequeños y aproximar la función *f* en cada intervalo por su valor en cualquiera de los puntos de ese intervalo (observa que eso es lo que se hace al definir las sumas inferiores y superiores, en las que se aproxima la función por sus valores mínimo o máximo en cada intervalo).

Si partimos el intervalo *[a,b]* en *n* "trocitos" usando la partición $\{a=x_0 < x_1 < x_2 < ... < x_n=b\}$, entonces en cada intervalo $[x_i,x_{i+1}]$ aproximamos la función *f*, por ejemplo, por $f(x_i)$. El área del rectángulo en este intervalo es entonces $(x_{i+1} - x_i) \cdot f(x_i)$ (base por altura). Para aproximar la integral, sumamos todas estas áreas (contando como negativas las que quedan por debajo del eje x). Por concretar, hagamos esto en el intervalo *[1, 3]*.

```
a = 1;  (* Extremo izquierdo *)
b = 3;  (* Extremo derecho *)
n = 50;  (* Número de trozos *)
h = (b - a) / n;  (* Paso del método *)
(* Definimos una lista con los puntos de la partición *)
y = Table[a + (b - a) * i / n, {i, 0, n}];
```

Una primera idea para calcular el área dada por los sucesivos tectángulos es usar el comando **Sum**.

```
Sum[h * f[y[[i]]], {i, 0, n - 1}]
```

El resultado es bastante curioso. Hemos hecho algo mal. ¿Qué? Observa la salida de la siguientes entradas.

```
y[[0]]
y[[1]]
y[[n - 1]]
y[[n]]
y[[n + 1]]
```

A partir de aquí es claro cuál ha sido el error cometido. Al crear la lista con los puntos de la partición hemos utilizado el contador **i** desde **0** hasta **n-1**. Sin embargo, cuando utilizamos los elementos de dicha lista hemos de tener en cuenta que empiezan en **y[[1]]** y acaban en **y[[n]]**. Por tanto, la manera coorecta de actuar es la siguiente.

```
aprox = Sum[h * f[y[[i]]], {i, 1, n}]
```

Comparemos esto con el cálculo que hace *Mathematica.*

```
NIntegrate[f[x], {x, a, b}]
```

■ Método del trapecio compuesto

En lugar de aproximar el área determinada por la función por medio de rectángulos, podemos aproximarla en cada intervalo por medio del trapecio que resulta de unir $(x_i,\ f(x_i))$ con $(x_{i+1},\ f(x_{i+1}))$. El área de un trapecio es (lado largo + lado corto)/2 * ancho, así que en cada intervalo aproximamos el área por $(f(x_i)+f(x_{i+1}))/2 * (x_{i+1}-x_i)$. Veamos cómo hacerlo.

```
a = 1;  (* Extremo izquierdo *)
b = 3;  (* Extremo derecho *)
n = 20;  (* Número de trozos *)
h = (b - a) / n;  (* Paso del método *)
(* Definimos una lista con los puntos de la partición *)
y = Table[a + (b - a) * i / n, {i, 0, n}];
```

Teniendo en cuenta el problema que surgió en el método visto antes, la aproximación se debe de calcular del siguiente modo.

```
aproxtrapecio = Sum[h * (f[y[[i]]] + f[y[[i + 1]]]) / 2, {i, 1, n}]
```

Como podemos comprobar, con este método obtenemos una mejor aproximación usando menos intervalos.

■ Método de Simpson compuesto

En lugar de aproximar el área determinada por la función por medio de trapecios, podemos aproximarla en cada intervalo por medio de la parábola que pasa por $(x_i,\ f(x_i))$, $((x_i+x_{i+1})/2,\ f((x_i+x_{i+1})/2))$ y $(x_{i+1},\ f(x_{i+1}))$. El cálculo del área de esta parábola justifica que en cada intervalo aproximemos el área por $(x_{i+1}-x_i)*(f(x_i)+4\,f((x_i+x_{i+1})/2)+f(x_{i+1}))/6$. A partir de la fórmula del trapecio compuesto podemos programar el método de Simpson compuesto.

```
a = 1;  (* Extremo izquierdo *)
b = 3;  (* Extremo derecho *)
n = 20;  (* Número de trozos *)
h = (b - a) / n;  (* Paso del método *)
(* Definimos una lista con los puntos de la partición *)
y = Table[a + (b - a) * i / n, {i, 0, n}];
aproxsimpson =
 Sum[h * (f[y[[i]]] + 4 f[(y[[i]] + y[[i + 1]]) / 2] + f[y[[i + 1]]]) / 6, {i, 1, n}]
```

A la vista de la salida que hemos obtenido, podemos conjeturar que *Mathematica* utiliza el método de Simpson compuesto cuando introducimos la orden **NIntegrate**.

■ Observación

Acabamos esta práctica con una observación que hemos de tener siempre en cuenta: una cosa es la teória y otra la práctica. En efecto, teóricamente nosotros podemos hacer que *n* sea tan grande como queramos. Sin embargo no es así cuando uamos un programa informático.

```
n = 2000;  (* Número de trozos *)
h = (b - a) / n;  (* Paso del método *)
(* Definimos una lista con los puntos de la partición *)
y = Table[a + (b - a) * i / n, {i, 0, n}];
aprox = Sum[h * f[y[[i]]], {i, 1, n}]
aproxtrapecio = Sum[h * (f[y[[i]]] + f[y[[i + 1]]]) / 2, {i, 1, n}]
aproxsimpson =
 Sum[h * (f[y[[i]]] + 4 f[(y[[i]] + y[[i + 1]]) / 2] + f[y[[i + 1]]]) / 6, {i, 1, n}]

n = 200000;  (* Número de trozos *)
h = (b - a) / n;  (* Paso del método *)
(* Definimos una lista con los puntos de la partición *)
y = Table[a + (b - a) * i / n, {i, 0, n}];
aprox = Sum[h * f[y[[i]]], {i, 1, n}]
aproxtrapecio = Sum[h * (f[y[[i]]] + f[y[[i + 1]]]) / 2, {i, 1, n}]
aproxsimpson =
 Sum[h * (f[y[[i]]] + 4 f[(y[[i]] + y[[i + 1]]) / 2] + f[y[[i + 1]]]) / 6, {i, 1, n}]

n = 2000000;  (* Número de trozos *)
h = (b - a) / n;  (* Paso del método *)
(* Definimos una lista con los puntos de la partición *)
y = Table[a + (b - a) * i / n, {i, 0, n}];
aprox = Sum[h * f[y[[i]]], {i, 1, n}]
aproxtrapecio = Sum[h * (f[y[[i]]] + f[y[[i + 1]]]) / 2, {i, 1, n}]
aproxsimpson =
 Sum[h * (f[y[[i]]] + 4 f[(y[[i]] + y[[i + 1]]) / 2] + f[y[[i + 1]]]) / 6, {i, 1, n}]
```

Desgraciadamente el programa "casca" al llegar a *n=2000000*. ¡Ojo! Este no es un problema exclusivo de *Mathematica*. Todos los programas caen antes o después ante uno de los principales problemas a la hora de hacer cálculos aproximados: **Los errores de redondeo**.

Ejercicios (Segunda parte)

1.- **a)** Dibuja la función $\sqrt{1-x^2}$ en el intervalo *[0,1]* (es un cuarto de circunferencia). Sin calcularla, ¿cuál debe ser su integral en ese intervalo?

b) Calcula su integral de forma exacta (simbólica), y luego calcula una aproximación numérica (con cualquier método adecuado).

c) Aproxima la integral anterior usando primero la aproximación por rectángulos y luego la fórmula del trapecio. ¿Qué aproximación es mejor?

d) Observa que puedes aproximar Pi con estos métodos numéricos. ¿Cuántos decimales correctos de Pi puedes conseguir de esta forma?

2.- Aproxima la integral de *f(x)=3xcos(x+1)* en el intervalo *[0,2]* usando la fórmula de los trapecios variando el número de intervalos desde 1 hasta 50. (Sugerencia: usa, por ejemplo, el comando **Do**).

3.- Se quiere calcular de forma aproximada el área de cierto terreno que se encuentra bordeado por un pequeño riachuelo. Para ello se han medido distancias al borde del mismo tomadas a partir de la valla que delimita dicho terreno y que sí está en linea recta. Las medidas en metros que se han obtenido, tomadas de diez en diez metros, son

0; 6; 10; 8; 5; 9; 12; 5; 0.

¿Puedes dar una estimación del área del terreno?